A PROPOSAL OF AN ARCHITECTURE FOR THE COORDINATION LEVEL OF INTELLIGENT MACHINES

by

R. Beard, J. Farah, and P. Lima

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering Department
Troy, New York 12180-3590

January 1993

CIRSSE REPORT #133

A Proposal of an Architecture for the Coordination Level of Intelligent Machines

Randal Beard, Jeff Farah, Pedro Lima

January 21, 1993

1 Introduction

This report is the result of several brainstorming sessions held during the Summer of 1992 which were extended through the Fall 92. In these sessions the authors addressed the issue of obtaining a practical, structured and detailed description of an architecture for the Coordination Level of CIRSSE Testbed Intelligent Controller. Previous theoretical[13, 25, 28] and implementation works[9, 17] were the departure point for the discussion.

The document is organized as follows: after this introductory section, section 2 summarizes our overall view of the Intelligent Machine (IM) as a control system, proposing a performance measure on which to base its design. Section 3 addresses with some detail implementation issues. An hierarchic petri-net with feedback-based learning capabilities is proposed. Finally, section 4 is an attempt to address the feedback problem. Feedback is used for two functions: error recovery and reinforcement learning of the correct translations for the petri-net transitions.

2 The Intelligent Machine as a Control System

The complete hierarchy of the Intelligent Machine (IM) may be seen as a control system with different levels of resolution. Control in this context means not only the servomechanism problem but also includes progressing

towards a goal (the reference) using feedback from the results obtained so far.

Viewing the IM as a control system requires the identification of the wide-sense *reference* and *feedback* signals. Using as an example the CIRSSE testbed case-study, the command Insert Strut is a *reference* or *set point* for the IM's organization level. The required output is obtained if the strut is correctly inserted. However, this implies that several internal wide-sense control loops worked equally well: at the execution level the compliant controller was able to deal with force errors without too much overshoot and the vision algorithm could deal with noise spots when getting frames of the strut fiducial marks; at the coordination level the strut was grabbed and the fiducial marks matched the internal pattern; and so on.

A failure to match the reference doesn't necessarily mean a crash or some damage. At the execution level, sensors may signal the eminence of a dangerous situation (example - near collision) or an event which jeopardizes the whole plan (example - gripper couldn't grab the strut). Thus, probabilistic convergence to 1 of the requirement that 'output matches reference' is satisfactory in most cases, since it is possible to anticipate and eventually recover from mistakes which may occur. In the cases where the error is not anticipated, a shutdown operation may be required.

Feedback flows bottom-up through the levels of the hierarchy. It has two goals:

- Feedback will be used in a *reinforcement learning*[1, 16, 21] mechanism to reward (penalty) (un)successful activities, primitive events and low level algorithms[1].

- Feedback provides information pertinent to the **Planning Coordinator**. This may be used to signal a failure and provide failure information. error recovering procedure.

A major concern of a control system designer is the *performance* measure for the controlled system. The design goal is to maximize that measure, or if the system is too complex, to provide the mechanisms that will let it learn how to maximize performance. In Intelligent Machines a possible

[1]When talking about activities, events and algorithms translating the events we follow the formalism of [24]

measure of performance seems is *entropy*[18].This result is due to the fact that entropy deals with information, independent of its source, whether that be an image to be processed by a vision algorithm, a coded control algorithm to be processed by a specific processor or the status information after the completion of a given task expressed as a predicate or collection of predicates. Actually, entropy measures *uncertainty* at all levels of the Intelligent Machine:

- at the **execution level**, there is uncertainty in terms of overshoots, position and velocity errors, rise-times and similar features. This uncertainty can be expressed statistically, hence entropies may be used. In citeMusto92 it is shown that, at least in some cases, feedback reduces the uncertainty about the above mentioned features thus reducing the entropy of the system. Other authors claim this is a general result, without actually proving it[29].

- at the **coordination level**, there is uncertainty in terms of the success of each of the *primitive events* composing an *activity*, as defined in [26]. At this level we deal with abstract features such as **strut grabbed**, **path planned**, **manipulator didn't move**.

- at the **organization level**, there is uncertainty in terms of the success of the *activity* planned.

In the sequel, we assume that for each event there is a initially assigned (at the design stage) set of algorithms capable of implementing the event, and the same happens with the assignment of events to activities.

If we minimize entropy, we are improving performance, since each of the actions at all levels will have a broader chance of success. This can be interpreted as a problem of improving *reliability*[12].:

- Given the maximum allowed entropy (minimum allowed reliability) required by the organization level for some primitive event (grab strut, plan path, move robot), the most *reliable* (i. e. the one which assures lowest uncertainty/highest probability of success) low-level algorithm must be chosen among the feasible ones for that event[13]. It is assumed that there exists at least one algorithm for each event.

- Given the maximum allowed entropy required by the command given to the organization level, the most reliable ordered sequence of primitive events (activity) must be chosen among the feasible ones.

However, the price to pay when a high reliability is required may be unacceptable, either in terms of computational time or resources such as memory, number of image frames, sampling rate.

The *ϵ-complexity* of a problem as defined in Information-Based Theory of Complexity[23] provides the lowest possible cost of an algorithm that solves the problem given some desired accuracy ϵ. The cost includes the prices of getting information and processing it. Depending on the model used, different features are weighted (CPU time, memory, etc). We are more interested in comparing the *costs* of different algorithms that are able to solve the problem, although complexity may immediately tell us that no algorithm exists that can solve the problem at hand with the required accuracy.

Given the above, the *performance* of the entire machine may then be formulated as an Optimization Problem:

$$\begin{array}{ll} \min & \text{cost}(a) \\ \text{s. t.} & \text{reliability} = R_d \end{array}$$

where R_d is the reliability required at each level by its hierarchic predecessor and a is an algorithm translation for an event or an activity translation for a command.

We may think of Reliability as

$$\text{P(worst case error with respect to specifications} < \epsilon) \tag{1}$$

for small ϵ and δ.

Cost is defined as

$$\text{cost}(a) = \sup_{f \in F} \{c_i(f) + c_p(f)\} \tag{2}$$

where f is a *problem element*, for example the overshoot of a control algorithm implementing a moverobot event, $c_i(f)$ is the cost of getting information about f and $c_p(f)$ is the cost of processing that information by algorithm a.

Finally, the error associated to a is, in a probabilistic setting

$$e(U) = \sup_{f \in F} \{\|f - \hat{f}\| : P(\|f - \hat{f}\| < \epsilon) \geq 1 - \delta\} \tag{3}$$

where $\|f - \hat{f}\|$ is the error between the desired f and the f obtained by algorithm a. Notice that F is a normed linear space. In simple words, this

means that we control the error of estimating f, keeping it below ϵ, except in a subset of F with measure δ. The cost is obtained from the constraints imposed on the error. For example, if we need to average N image frames to reduce the error of locating an object below ϵ, and if we don't consider the cost of processing that information, the overall cost will be equal to c_i and thus proportional to N.

Equations 1 and 3 are similar, if we consider f as a vector of specifications for a given problem. Hence, the formulation of Information-Based Theory of Complexity seems appropriate to help us in the design of the IM, expressed by the optimization problem above, if we make $R_d = 1 - \delta$[11].

3 The Coordination Level

The purpose of this section is to propose an architecture for the Coordination Level which incorporates some of the ideas outlined above. Some of the specifications for such an architecture are outlined below.

1. The Coordination Level should receive one activity from the Organization Level and translate it into a sequence of low level algorithms which can be executed by the Execution Level. It should then schedule and execute these algorithms, coordinating their interaction in a deterministic manner.

2. The Coordination Level should have the ability to choose among alternative algorithms, if available, for each event, based on their reliabilities and costs, as described in the previous section.

3. Cost measures which map activity translation proposed by the Coordination Level onto the real line need to be developed.

4. The Coordination Level should have decision capacity to choose among alternative translation proposals, thus enabling the Coordination Level to optimize these cost measures in some sense. The cost measures should depend on the reliability and complexity of the algorithms at the Execution Level, and also on the efficiency of distributing these routines on the underlying parallel hardware.

5. The Coordination Level should have a mechanism of learning the reliability parameters of each algorithm, based on the feedback from the execution level.

6. The Coordination Level should provide boolean feedback to the Organization Level indicating the successful/unsuccessful application of an activity.

7. Feedback and stability *within* the Coordination Level need to be defined.

8. The Coordination Level should contain an error recovery mechanism.

3.1 Proposed Architecture for the Coordination Level

In the following we will propose an architecture which addresses these specifications. The structure which we propose is an extension to that given in [18] [19] [28]. We propose that the Coordination Level be organized as shown in Figure 3.1. A brief explanation of the flow of information in Figure 3.1 will now be given.

The purpose of the **Organization Level** is to receive and interpret a user command and then formulate a sequence of commands that will achieve the requested job [25]. The Organization Level is designed to organize a sequence of abstract actions from a set of high level primitives which are stored in the long term memory of the machine. Various Artificial Intelligence techniques can be used to reason, plan and make decisions about the sequence of events which is the most likely to achieve the particular command issued by the user [18]. The information which flows along **arc 1** in Figure 3.1 is a string of events which represent a "task level" description of the task to be performed by the machine. The Organization Level receives boolean feedback information via **arc 4**, which represents the success or failure of certain activities. Information regarding the state of the system when success or failure occurs, is not provided to the Organization Level.

The Translator receives a string of commands from the organization Level. This string is then translated into a Petri Net which will execute the string of commands issued from the organizer. The translator is intelligent

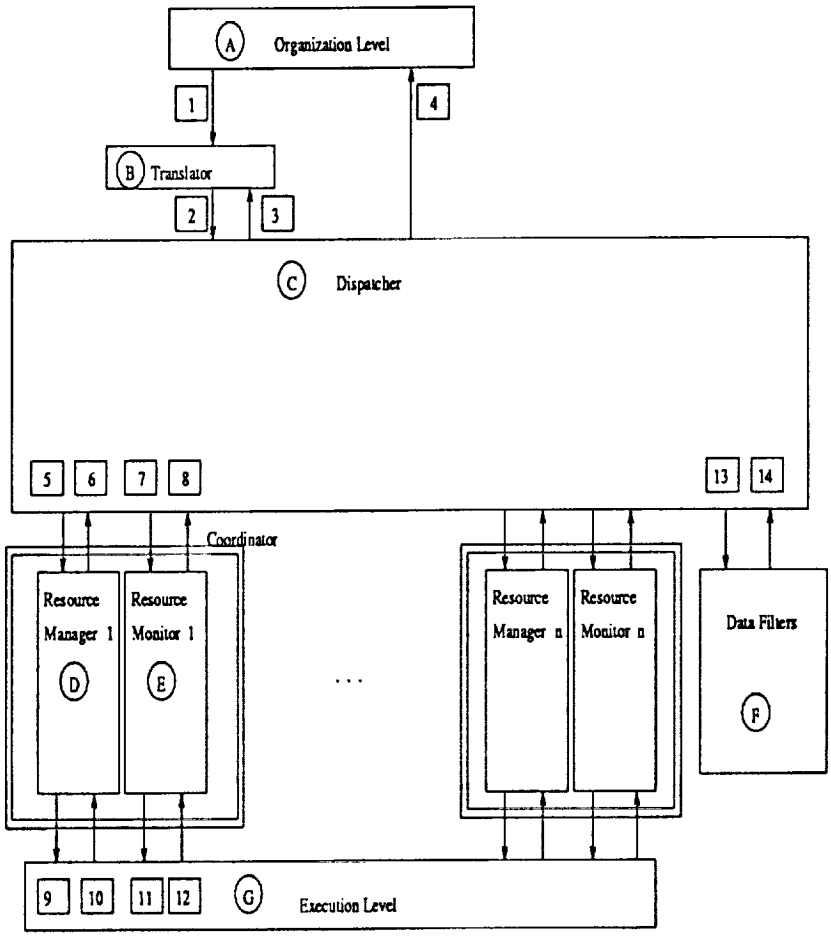

Figure 1: Flow Chart of the Coordination Level.

in that it must perform two tasks requiring intelligence. These two functions are;

1. the translator must decide which dispatcher primitives will be most likely to achieve the commands issued by the organization Level, and

2. the translator must impose ordering constraints upon the primitives determined in 1.

The first of these functions could be implemented with a neural network, and the second function could be solved using combinatorial optimization approaches like genetic algorithms or simulated annealing. The objectives of the translator are to produce a Petri Net which accomplishes the task received from the Organization Level, and optimizes the *a priori* estimate of the cost measures associated with that Petri Net. The translator sends the dispatcher a description of the high level Petri Net which it has produced via **arc 2**. The translator receives feedback from the dispatcher via **arc 3** This feedback is in terms of the success of the *events* associated with the previous translation. If the previous translation resulted in a successful control system then the parameters of the translator are positively reinforced.

The dispatcher is a feedback controller for a distributed discrete event dynamic system. It is implimented with Petri nets which are derived by the translator. The search space over all possible Petri nets is inhibitively large, therefore it is desirable that it be reduced by limiting the Petri Nets considered by the translator. Two obvious restiction is that the Petri Nets must be live and bounded. We will therefore impose structural constraints, which ensure liveness and boundedness on the Petri Nets which are derived by the translator. Imposing these constraints produces two desirable effects; the space which the translator must search to find a feasible controller is greatly reduced and we eliminate the necessity to test the controllers properties using Petri Net tools such as GreatSPN [2] or SPNP [3]. A description of the class of Petri Nets proposed for the dispatcher is briefly outlined in section 3.2, and will be analytically defined and shown to be live in future work. The dispatcher communicates with the coordinators via **arcs 5-8** The dispatcher sends the resource manager requests for a particular procedure to be executed. These procedures have data requirements. For example, movePathRobot() requires that the path along which the robot will move is

specified. As will be explained in section 3.2 some of the tokens in the dispatcher carry pointers associated with data. When a procedure is requested from a coordinator, pointers to the input data required for that procedure are also passed to the coordinator. When the requested procedure is completed the resource manager notifies the dispatcher and sends pointers to output data produced by the procedure back to the dispatcher. For planning and reasoning purposes it is imperative that data be represented explicitly at the dispatcher level, however, communication constraints require that a minimum amount of information flow between the dispatcher and the coordinators. Therefore pointers to data offer a suitable compromise. These pointers contain information regarding the machine and the memory location on that machine at which the data resides.

The **Coordinators** are broken into two separate functional units; the **Resource Manager** and the **Resource Monitor**.

The **resource manager** is a software process which is established when the intelligent machine is initialized. It guarantees that two primitives associated with that resource are not executed simultaneously. For example openGripper() and closeGripper() cannot be executed simultaneously since they are executed by the same physical hardware. The role of the resource manager is to process requests from the dispatcher, place these requests in a priority queue and execute the requests sequentially by priority. A resource should be established whenever there is more than one hardware primitive which requires the same piece of hardware. Therefore a "motion coordinator" would be replaced by "right arm coordinator" and "left arm coordinator," or even "third joint coordinator" depending on how the controllers are implemented.

The resource manager is not a Petri Net, but should be modeled as a Petri Net to analyze the effect that interfacing to it has on the properties of the dispatcher level Petri Net. The Petri Net model of a resource manager is shown in Figure 3.1. A token in a place RP_j is a request for a particular primitive to be executed. The introduction of the resource manager makes it impossible to guarantee that the Dispatcher is bounded. However, this is a property can be used to our advantage by using it to define a cost measure for the system.

The **resource monitors** are processes which are also established when the intelligent machine is initialized. Resource monitors initialize and maintain a group of monitoring algorithms which periodically check that certain state variables satisfy specific constraints. A predicate value is associated

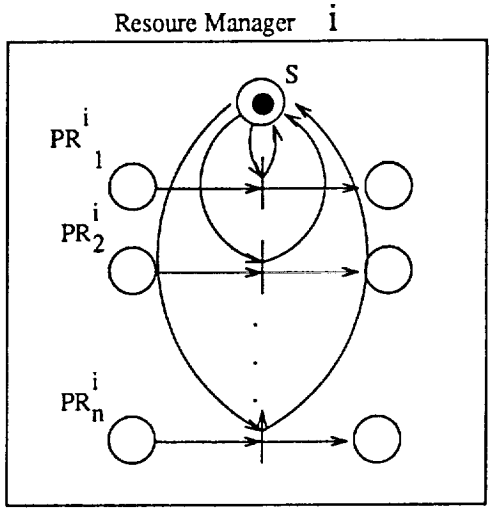

Figure 2: Petri Net model of resource manager.

with each state variable. This predicate variable will initially take values of TRUE/FALSE. The dispatcher will query the resource monitor concerning the value of these predicates and the resource monitor responds with the appropriate value. Monitoring algorithms are the key element in the definition of the Coordination Level internal feedback structure which will be defined in future work.

The dispatcher is also able to spawn **Data Filters**. These are processes which are not associated with any particular hardware resource, and can therefore be run on the machine which is the least busy at that time. For example once a digital image has been obtained with a camera, there may be a program which analyzes the image and produces the location of an object. This program is a data filter and does not have any specific hardware requirements, and can therefore be run on any machine in the network. Therefore a resource manager is not required to schedule execution of these programs. Data filters receive input data pointers from the dispatcher and return output data pointers via **arcs 13** and **14**.

Testbed Hardware Routines are routines that interface directly to the testbed hardware. They are precompiled routines that are spawned and

executed by the coordinators. Resource managers establish processes via **arcs 9** and **10** which run testbed hardware. These processes control and manipulate the hardware. Resource monitors establish hardware routines which test the status of various processes and hardware via **arcs 11** and **12.**

3.2 Petri Net Structure Within the Coordination Level

The Petri Nets associated with the dispatcher fulfill two distinct functions

- First, they establish and maintain the process flow, by establishing precedence relations which are inherently dictated by the task under consideration. For example, we must move the end effector close to an object before we can grasp it, etc.
- Second, they establish data flow in the system. For example data produced by a path planning algorithm is sent to the motion control system which actually moves the arm along the specified path.

These two functions are distinct and should be represented explictly in the Petri Nets comprising the dispatcher. Therefore the dispatcher will be a superposition of two Petri Nets, one ensuring the integrity of process flow and the other ensuring the integrity of data flow. Corresponding to these two nets will be two types of tokens which are attached to pointers that represent certain information. *Process tokens* will carry information about the current process which is being executed. *Data tokens* will carry data pointers which are needed to execute the data filters spawned by the dispatcher. The Petri Net which makes up the dispatcher will contain two types of places and three types of transitions. *Process places* contain process tokens and *data places* contain data tokens. Process places and data places are disjoint. The three types of transitions are as follows. *Process flow transitions* simply specify the flow of process tokens through the net. *Hierarchical transitions* spawn and execute processes. These process may either be lower level Petri Nets or requests to a resource manager to execute a hardware routine. Both process places and data places can be connected to hierarchical transitions. *Data processing transitions* execute data filters. Only data places are attached to these transitions.

The dispatcher is organized as a hierarchy of Petri Nets. It is common practice to organize Petri Nets in a hierarchical structure for modeling purposes [5] [22] [27]. However when a controller is actually implemented from this Petri Net the "flat," or expanded Petri Net is actually executed [17]. This results in several problems. One such problem is that it becomes difficult to execute complex tasks which are represented by extremely large Petri Nets. When the Petri Net becomes large, the time spent searching for enabled transitions becomes very large. Various tricks such as random linear search [17] and object oriented techniques are used to improve the search time. However these tricks can result in starvation [14]. Therefore it is desirable to only execute a small portion of the net at a time. By dynamically executing the Petri Net in a hierarchical fashion this problem is eliminated. Each lower level Petri Net is executed as a separate process which is spawned when the transition representing that Petri Net is fired, therefore only a small portion of the transitions are being searched at any one time, and these transitions are limited to those which are most likely to be enabled at that instance of time.

3.3 Translation of an Event into an Algorithm

Some additional considerations about the translation of an event into a Testbed Hardware Routine by means of a Hierarchical Transition need to be expande upon. Each event must be translated into a low level algorithm which accomplishes the respective task (examples: moverobot, locateobject, planpath. The system has a set of algorithms capable of implementing that event. In each instance, a set of *feasible algorithms* is determined given the present states of the system and of the environment. The set of feasible algorithms is composed of:

- Algorithms associated with the event that, according to some internally stored and regularly updated *world model*, are appropriate for the current state of the world. Hence we get rid of algorithms that are not suited for the current conditions, even though they are able to solve the problem associated with the event,
- Algorithms associated with the event that have a current estimated reliability greater than or equal to the desired reliability assigned to the event by the Organization Level.

The algorithm of least cost among all the feasible algorithms is selected and applied to solve the problem associated to the event. The model underlying the cost computation reflects the main issue in terms of performance for that event: memory, CPU time or other(s)[23, 11].

Initially, the estimated reliabilities of the events composing the activity may all be set to the same value. The learning mechanism will essentially learn the nominal reliability of each algorithm along time, as the algorithm fails or succeeds accomplishing its assigned tasks.

According to what was proposed in the previous section, each primitive event should thus have associated with it a corresponding resource primitive. One of the rules of the resource primitive is to choose among different possible algorithms feasible for that event the one of less cost. This resource primitive should also implement the feedback algorithm for the event, rewarding algorithms that were successful implementing the event and penalizing the unsuccessful ones.

3.4 Learning the Reliability Parameters

At the coordination level, the algorithm parameters referred to in the last section must be updated in time in order to reflect the increase (decrease) of confidence in a given action after its (un)successful application to a particular state of the world.

Reliability as a statistic measure of the success of control/vision algorithms for pose estimation was studied in [12]. Probability distributions are used to model the uncertainty associated with the success of applying an action to the controlled system. In this context, success means that specifications were met with some pre-assigned accuracy ϵ (see equation 1). However, some parameters are assumed as previously known, such as the mean and variance of the gaussian distribution for the maximum overshoot or final position error. The learning procedure may change these parameters in a suitable way. For instance, if the low level control fails due to an unexpected oscillation, the mean value of the overshoot must be increased for that particular algorithm, thus reducing the estimated reliability of getting a low overshoot with that particular algorithm in that particular world. This will discourage further applications of the algorithm when overshoot is critical.

4 The Feedback Hierarchy

Wide sense feedback is present now only at the Execution Level of the Intelligent Machine, either in control or vision algorithms (ex. determine the position of an object using a reference prototype in an internal data base). It seems intuitive that feedback from one level to its hierarchic predecessor will reduce costs (for example the best path may be learned by trial, error and reward along time[10]), improve reliability (the more accurate we are about the reliabilities of the different algorithms, more chances of success we have for the entire activity) and provide error recovery.

Feedback may be used for two distinct functions:

- A *reinforcement learning* scheme will reward/penalize the set of activities, primitive events or low level algorithms which induced some state (marking of a petri-net, for ex.) in the level hierarchically below. The amount of reward/penalty will depend on a classification based on error analysis.
- In *error recovery*, procedures will be assigned, depending on the failure classification.

4.1 Feedback for Reinforcement Learning

In simple terms, the flow of feedback for reinforcement learning through the hierarchy is:

- at the **coordination level**, the association of a low level algorithm to a primitive event is rewarded/penalized after the completion of the last chosen low level routine for that event. The amount of reward/penalty depends on the state of the execution level after the application of the event. Failures and successes of a plan should also modify the importance of each event inside the activity.
- at the **organization level**, one activity is rewarded/penalized after the completion of the corresponding ordered sequence of events. The amount of reward/penalty depends on the state of the coordination level after the application of the activity.

More work needs to be done in this area, in order to clarify, implement and formalize concepts.

4.2 Error Recovery

The Planning Coordinator is a logical extension to the Coordination Level of the Intelligent Machine Model, functioning to provide a heretofore unavailable platform for robust error recovery and dynamic on-line planning by autonomous and semi-autonomous robotic systems. This section introduces the design architecture of the Planning Coordinator and focuses upon its macrostructure, interfaces and functional description, with respect to its role as the mechanism whereby an existing robotic system requiring significant human intervention can be made more autonomous, thus becoming more robust.

4.2.1 Planning Coordinator: Macro Architecture

Physically, the Planning Coordinator is a functionally stratified, stand alone device operating as a logical extension to the Coordination Level of the Intelligent Machine. It logically connects to the physical communication scheme of the Intelligent Machine through external communication ports. Unlike the other Coordinators in the Intelligent Machine, the Planning Coordinator does not communicate directly to any piece of physical hardware or to the Organizer. To engage in communication, the Planning Coordinator utilizes the communication schemes that exist between the other Coordinators and the hardware they coordinate, as well as the communication scheme between the Dispatcher and the Organizer. As a logical extension to the Intelligent Machine, the Planning Coordinator is logically subservient to whatever is considered to be the main controller of the Intelligent Machine. This is necessary to ensure system coherence. In the event of a catastrophic main controller failure the Planning Coordinator might assume the role of the overall system controller, but to a very limited extent. The Planning Coordinator is expanded in *Figure* 3, to introduce the constituent parts of its macro architecture, listed below. These parts are grouped by level.

Current World Model (CWM) - Level 1 Shadow Coordination Level Petri Net (SCPN) - Level 1 Primitive Structure Database (PSDB) - Level 2 Node/Link Weighting Mechanism (N/L-WM) - Level 2 Mapping Mechanism (MM) - Level 2 Error Recovery Generation Algorithm (ERGA) - Level 2 System Fault Monitor (SFM) - Level 3

With the exception of the System Fault Monitor, these components constitute the mechanisms whereby a task level error recovery (or on-line plan

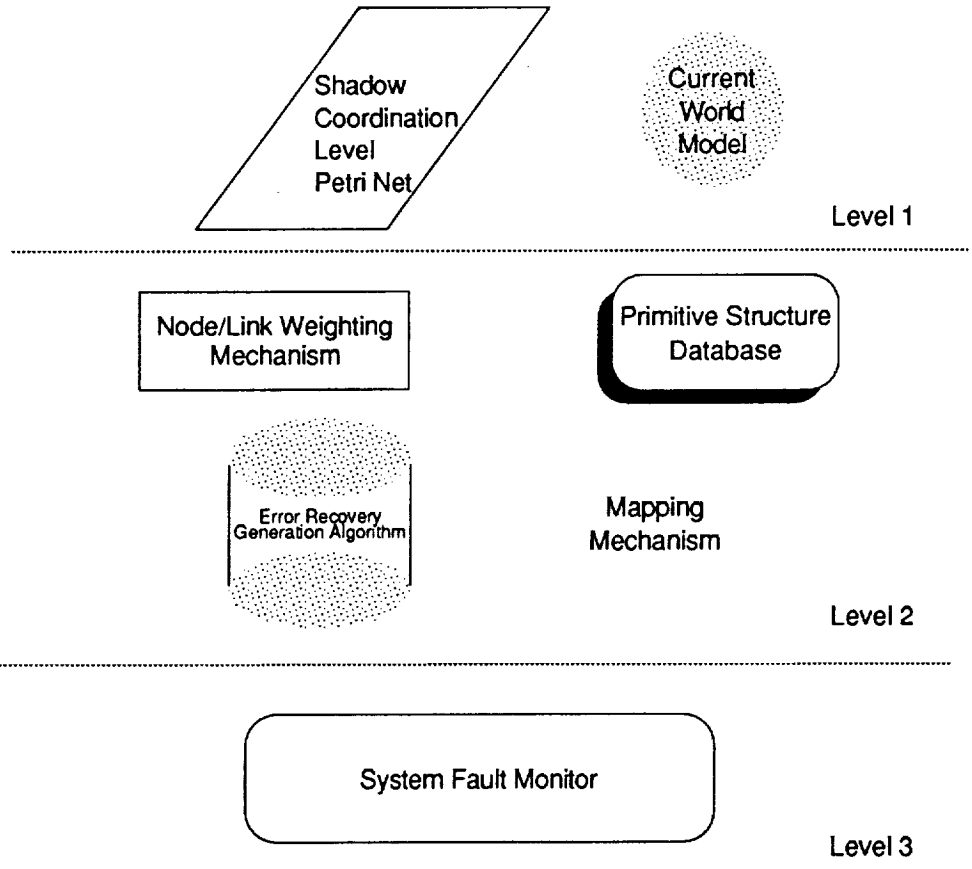

Figure 3
The Planning Coordinator (PCOORD) Constituent Parts

which henceforth is considered to be a specific instantiation of an error recovery), will be successfully executed. The following subsections elaborate on the seven constituent parts that comprise the Planning Coordinator. Of highest significance to this document are the five major component parts of the Planning Coordinator: *Current World Model, Primitive Structure Database, Node/Link Weighting Mechanism , Mapping Mechanism, and Error Recovery Generation Algorithm.*

4.2.2 The Current World Model

The Current World Model is a dynamically changing, linguistic representation of the most current environmental information available to the Planning Coordinator. Its function is to accurately represent the most current status of the environment in which the Planning Coordinator, and hence the Intelligent Machine, must operate. Unlike the long term, quasi-static Global World Model, the Current World Model maintains a shorter term representation. With two major exceptions, only a portion of the Current World Model will be active at any one time. The first exception occurs when information from the Current World Model is initially interpreted to create the Primitive Structure Database. The second exception occurs when the Current World Model is called upon to update the Global World Model with new information derived from the activities of the Planning Coordinator.

Note that in the development of the Planning Coordinator, it is anticipated that the Current World Model will change significantly over a long period of time and as such, differ significantly from the Global World Model. A question arises as to the need for the Current World Model if a Global World Model exists and can be made to be accessed reliably and efficiently. This question is answered as follows. The Primitive Structure Database, to be discussed, represents Primitive Structures derived from the Current World Model. When the Current World Model changes, the existing Primitive Structures are not lost. They remain in the Primitive Structure Database which is augmented through the addition of the new primitive structures. Hence were the Global World Model to succumb to a catastrophic failure, none of the information that had been contained in it would be lost. This is because the Global World Model can be regenerated from the information stored in the Current World Model and the information stored in the Primitive Structure Database. The regeneration of the Global World Model

from the Current World Model and the Primitive Structure Database is not of concern to this document. It is considered to be potentially future work outside of the scope of this document, and is of itself a research area.

4.2.3 Shadow Coordination Level Petri Net

Within the hierarchy of the Intelligent Machine Model, task representation is performed through the use of Generalized Stochastic Colored Petri Nets as described previously. To maintain the continuity of this representation during its design phase, the Planning Coordinator utilizes a Shadow Coordination Level Petri Net, as a graphical representation tool. It is anticipated that the Shadow Coordination Level Petri Net will eventually become unnecessary and will be eliminated as a graphical representation tool. However, its function will be maintained.

The Shadow Coordination Level Petri Net is functionally, an exact copy of the executing Coordination Level Petri Net generated by the Dispatcher of the Coordination Level of the Intelligent Machine. Depicted in *Figure* 4a is a Coordination Level Petri Net, and in *Figure* 4b, the Shadow Coordination Level Petri Net, identified by its transitions. These transitions maintain connections to Map Interface Error Recovery Nodes that reside within the Mapping Mechanism of the Planning Coordinator. Through these connections it is possible to determine the exact location from which an error recovery would be enacted should the need for one arise, since errors occur only at Petri Net transitions. Identifying the locations of potential errors permits the pre-creation of most likely errored event recovery plans. Hence when an error does occur, and is the most likely errored event, an immediate response is possible. Considering an error that is not the most likely errored event requires the use of alternate plans. These alternate plans are built up from Primitive Structures that represent the actions and objects existing in the environment of an intelligent machine. They can be stored in a database for retrieval. This database is called the Primitive Structure Database and is the subject of the next subsection.

4.2.4 The Primitive Structure Database

The Primitive Structure Database (PSDB) is a database containing Primitive Structures that represent the basic operations that can be performed by

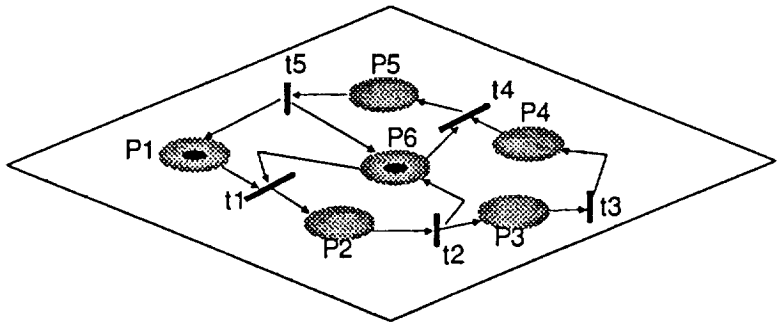

Figure 4A
A Coordination Level Petri Net

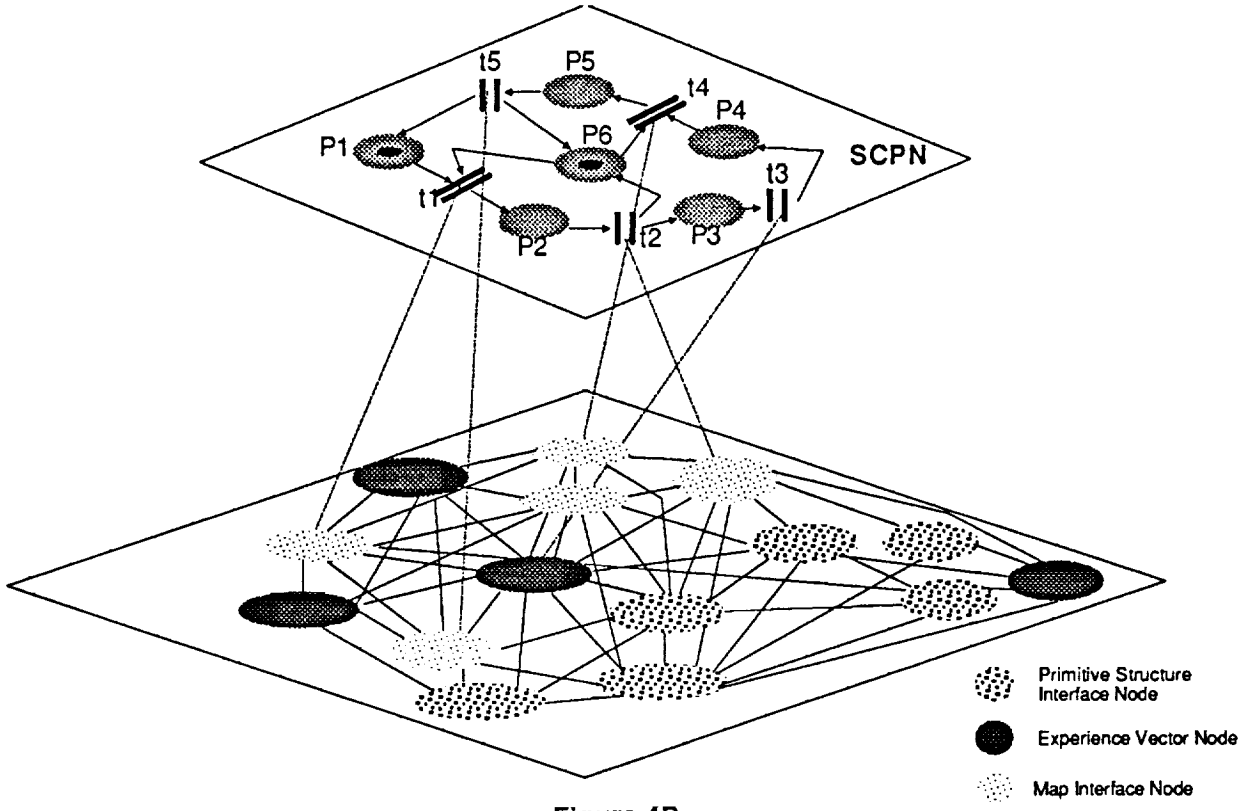

Figure 4B

The Shadow Coordination Level Petri Net Equivalent

an intelligent machine, as well as the objects that exist in the environment of an intelligent machine. These Primitive Structures are derived from the Current World Model which represents the most up to date environmental information available to the intelligent machine. The formal definition of a Primitive Structure is given below:

Primitive Structure

A potentially complex, building block created by the Planning Coordinator, based upon environmental information contained in the Current World Model, and functioning to represent the primitive actions (objects) capable of being performed (identified) by the intelligent machine. Several Primitive Structures can be combined to form complex plans that can later be used as either error recovery plans or on-line plans.

In keeping with the general structure of the Intelligent Machine Model, the Primitive Structures are individually, live, safe, bounded Petri Nets. Synthesizing these smaller Petri Nets into larger ones has been shown by Zhou, DiCesare, Narahari, and Koh to result in live, safe, bounded Petri Nets given that the properties of liveness, safeness, and boundedness existed in each of the smaller Petri Nets [30] [4] [15] [6]. In Section 2, both Generalized Stochastic Colored Petri Nets and Semantic Networks were introduced. The following builds on the descriptions of these two constructs.

The Primitive Structure Database is modeled using a Semantic Network. The Semantic Network model makes use of nodes, representing events and/or objects, and directed arcs, representing the relationships between objects. The nodes can be hierarchically structured, thus permitting descendants to inherit form and function from their ancestors. This is important because the descendants themselves may be separate nodes in the PSDB. In addition, through the use of case-frames as previously described, inherent search limiting agents are built into the Primitive Structure Database nodes. Finally, each of these nodes is connected in some way to other nodes. The connections may be simple or multiple, depending upon the complexity of the Primitive Structure Database. These connections are achieved through the use of linguistically identified, directed, relational arcs. The relational arcs permit conceptual relations between the nodes and as such are the natural points at which the strengths of such relations can be established. Since the relational links are directed, they provide natural pathways from one node to another. These natural pathways can be exploited to establish ordered error recovery or on-line plans. There is a difficulty in using this approach,

however. There may be multiple paths between any two nodes in a network. To assign a strength value to each of the links between any two nodes and to distinguish between the multiple paths that may exist between any two nodes (i.e., choose the best path from among all), the Node/Link Weighting Mechanism was introduced. The Node/Link Weighting Mechanism, is the subject of the next subsection.

4.2.5 The Node/Link Weighting Mechanism

The Node/Link Weighting Mechanism is one of the five major components of the Planning Coordinator. It functions to assign fuzzy weights (f-weights) to the nodes and relational links that comprise the SNet based Primitive Structure Database. The f-weights are used for two purposes. The primary purpose is to establish the relational strength(s) of one node to another, based on the linguistic relation connecting them. The secondary purpose is to combine the f-weights assigned to each individual relational link in some established path plan, and defuzzify the result. The defuzzified result provides the overall possibility of success number that the plan's path represents. The possibility of success number is then used to hierarchically organize, from highest possibility of success to lowest possibility of success, all of the plan paths whose possibility of success numbers' exceed some established threshold value. This organized list, called a plan execution list, contains those plans that have been deemed applicable to some error recovery or on-line plan request.

Once a possibility of success number has been determined, each of the relational links along the plan path that is responsible for the number's creation is assigned an ordering identifier, indicating which plan or plans in the execution list, the link refers to. The assignment of the relational f- weights, the combining of a series of relational f-weights into an overall plan f-weight, and the defuzzification of the overall plan f-weight into a 'crisp number' are examined in the following subsections.

4.2.6 Assignment of Relational F-Weights

The universe of discourse represented in an intelligent autonomous system is that derived from the system's knowledge of its environment as transformed into Primitive Structures and the relationships between Primitive Structures.

As has been stated, the Primitive Structures are maintained in a non-fedback *SNet* which is a structure very similar to the Fuzzy Cognitive Map of Kosko and Styblinski [7] [20]. These similarities permit the use of Fuzzy Cognitive Map techniques.

In establishing the Primitive Structure Database the first step is to derive the Primitive Structures and the relationships between Primitive Structures from the environmental information. Here it can be assumed that the Primitive Structures have been made available with no loss of generality. The resulting universe of discourse is an arbitrarily large but finite set of interconnected nodes, similar in structure to the Dempster-Shafer frame of discernment [8]. The major difference results from the fact that the universe of discourse is dynamic and hence it is necessary to use a Semantic Network base which permits changes in the base without destruction of the existing base.

The Node/Link Weighting Mechanism utilizes a dynamic Fuzzy Rule Base created by an Expert System from the available environmental information. Prior to the operation of an *x-autonomous system*, there are basic rules available, akin to the instinctual capabilities that human beings possess from birth. As the *x-autonomous* system begins to operate, its environment changes and the Expert System derives new rules to be added to the dynamic Fuzzy Rule Base. The feasibility of the dynamic Fuzzy Rule Base has been demonstrated [7]. The Primitive Structure Database, is initially a semantic network. Until application of the Node/Link Weighting Mechanism, there are no f-weight relations. The Node/Link Weighting Mechanism takes two connected nodes in the Primitive Structure Database and their linguistic relational arc and applies them to the Fuzzy Rule Base. The result of applying these two nodes and the arc is an f-weight, which is applied to the arc. When two nodes have multiple connections, potentially differing f-weights are assigned to each of the individual relational arcs. In this way, two nodes can have varying degrees of relational strength, based on the relation itself. This same procedure is applied to all pairwise nodes in the Primitive Structure Database according to the general procedure outlined below.

General Procedure

(Prior to beginning x-autonomous system operation)

Step 1: From information in the Current World Model, use Expert System to derive dynamic Fuzzy Rule Base.

Step 2: From Current World Model derive the nodes and links for primary

semantic network.

Step 3: Beginning from any node in the resulting connected digraph, utilize minimal time, complete search techniques to search and mark the entire digraph. During marking, take any two connected nodes and the directed arc between them and apply them to the Fuzzy Rule Base, resulting in an arc f-weight.

Step 4: Apply the f-weight to the relational link and return to Step 3 until the entire digraph is done.

(Upon completion of the Node / Link Weighting)

Step 5: As new nodes are added to the newly created Primitive Structure Database, begin at a newly introduced node and determine which node or nodes it is connected to. Apply the newly introduced node and the nodes it is connected to, to the Fuzzy Rule Base and determine an f-weight. Apply the f-weight to the new relational link as before.

End Procedure

Utilizing the above permits the establishing of a new Primitive Structure Database, or the augmenting of an existing Primitive Structure Database. As has been described, it is possible to start at one node (i.e., a start node) and efficiently trace out a path or paths to another node (i.e., a destination node). It is likely that with the high probability of multiple connections existing between two nodes, there will be multiple paths between two nodes. Within the confines of error recovery it is necessary to differentiate these paths into a hierarchically ordered plan execution list. This requires the combining of individual link f- weights into an overall plan f-weight, the subject of the next subsection.

4.2.7 Determining Overall Plan F-Weight and Creating Plan Execution List

Once the Primitive Structure Database has been constructed it is ready to be used by the Planning Coordinator for error recovery. In its final form, the Primitive Structure Database resembles a Fuzzy Cognitive Map. This resemblance permits the application of Fuzzy Cognitive Map Summation Methods to sum the individual link weights along a particular path. Once the weights along a particular path have been summed, the overall value is defuzzified into a crisp number. This crisp number is then used for comparison against a threshold value. If the number exceeds the threshold value, the plan is con-

sidered viable and is placed in the plan execution list. The general procedure is given below:

Plan Execution List Generation

Step 1: Identify plans generated by Primitive Structure Database search.

Step 2: For each plan, start at the start node and trace plan through to the destination node. At each connecting link, assign plan identifier to each link along the plan path, and store each link fuzzy value.

Step 3: For each traced through plan, take link fuzzy values and apply Fuzzy Cognitive Map Summation Method to it. Defuzzify the result into a crisp number. If the crisp number represents a value that exceeds the established threshold value, store the value and the plan identifier in the Plan Execution List, else discard it.

Note: If crisp number represents first plan, store in first slot of Plan Execution List regardless of its value. This will ensure that at least one plan is available to be tried. For each subsequent plan entered into the Plan Execution List, use binary search to find the correct position for the new plan in the Plan Execution List. If a subsequent plan values exceed the first plan value, but do not exceed the threshold, then replace the first plan with the subsequent plan.

Step 4 : Update link identifiers in Step 2, to reflect plan position in the Plan Execution List of Step 3.

Note: It is possible that a single link may be needed for more than one plan. Due to this possibility, a single link may have multiple identifiers.

End Procedure

The result of this general procedure is a Plan Execution List containing ordered viable plans, with ordering numbers. These ordering numbers are used to execute the plans sequentially until one plan is successful or until all plans have been exhausted. If a plan is executed and is successful, the plan is rewarded. If a plan is unsuccessful, it is penalized. The method of applying a reward or penalty is as yet undetermined and remains an open area for further research. Previously, it was stated that through the Shadow Coordination Level Petri Net it is possible to immediately identify where an error recovery must begin and end. This is due to the fact that errors can occur only at the Petri Net transitions. These transitions are connected to Map Interface Nodes that reside on *Level 2* of the Planning Coordinator. The following section introduces the Map Interface Nodes as well as the Mapping Mechanism of the Planning Coordinator.

4.2.8 Mapping Mechanism

The desired one-to-one mapping mechanism of the Planning Coordinator is both a structure to maintain three specific node types and a methodology to efficiently perform the following three functions:

1. Connect Shadow Coordination Level Petri Net transitions to *Map Interface Nodes.*
2. Connect *Primitive Structure Interface Nodes* to their Primitive Structures in the Primitive Structure Database.
3. Create *Experience Vector Nodes* based upon previously enacted, successful error recoveries.

The three different node types are defined below.

Map Interface Node

A dynamically allocated, two or three port active, interface point that connects to a Shadow Coordination Level Petri Net transition, the start (end) node of an error recovery or on-line plan, and an Experience Vector Node. Two Map Interface Nodes are created for each Shadow Coordination Level Petri Net transition. They are created at the same time as the Shadow Coordination Level Petri Net.

Primitive Structure Interface Node

A Primitive Structure Interface Node represents a pointer to a Primitive Structure in the Primitive Structure Database. It is necessary that each Primitive Structure be represented by a Primitive Structure Interface Node to ensure the identification of the start and end nodes of an error recovery or on-line plan.

Experience Vector Node

An Experience Vector Node is attached to the third port of the input Map Interface Node (i.e., the side connected to the input side of the Shadow Coordination Level Petri Net transition). It represents a successfully enacted error recovery or on-line plan sequence. The Experience Vector Node maintains the entire plan path through a vector of Primitive Structure Interface Node identifiers. These identifiers maintain the order of execution of the nodes. In addition, the Experience Vector Node maintains the state of the system when the error occurred. This permits an immediate response to an *identical* error.

Note that if the same error recovery does not work for an identical error, the Experience Vector Node is updated with information on the new error solution, when the new error solution is found.

In order that a Map Interface Node can connect to any Primitive Structure Interface Node, as is required for the operation of the Planning Coordinator, the network of Map Interface Nodes and Primitive Structure Interface Nodes must be a fully connected network. The choice of which links in the Map Interface Node / Primitive Structure Interface Node network to make active and which to leave inactive is determined by the places in the Shadow Coordination Level Petri Net that immediately precede and follow a transition connected to a Map Interface Node.

A considerable amount of the work that the Planning Coordinator must do can be done while the overall Intelligent Machine is not engaged in error recovery. As a result, the preprocessing of the error recovery plans can be done in parallel with the execution of the Intelligent Machine, saving overall execution time. This does not mean that the Planning Coordinator's primary function as an error recovery unit remains dormant until the preprocessing is done. The Planning Coordinator's overall function is governed by the Error Recovery Generation Algorithm, the subject of the next subsection.

4.2.9 Error Recovery Generation Algorithm

The Error Recovery Generation Algorithm governs the operation of the Planning Coordinator. The algorithm itself assumes the availability of specific information from the Intelligent Machine, the accessibility of communication ports to the Intelligent Machine and priority over all other coordinators. Some of the information which must be provided by the Intelligent Machine to the Planning Coordinator includes.

* Error status based upon a flag called ERR-FLAG. If asserted, an error is present. If not asserted, no error is present. * On-line Plan status based upon a flag called OP-FLAG. If asserted without ERR-FLAG, a short term on-line plan is required. If asserted with ERR-FLAG, an interactive on-line plan is required. * Intelligent Machine main controller status based upon a flag called MC-FAIL. If asserted, the main controller has failed.

Figure 5 shows the two stage flow diagram of the operation of the Error Recovery Generation Algorithm. Stage 1, details the preprocessing of the initial Current World Model, Fuzzy Rule Base, and Primitive Structure

Database. Stage 2, details the preprocessing of the initial error recovery routines and the processing of error recoveries, on-line plans, and modifications to the Primitive Structure Database. The steps involved in the Error Recovery Generation Algorithm are detailed following the figure.

Prior To Commencing Operation of the Intelligent Machine

Step 1 ¿From the Global World Model create the Current World Model. Initially, the Current World Model and the Global World Model will be the same.

Step 2

A) From the Current World Model, use an Expert System to create the dynamic Fuzzy Rule Base. Initially this base may contain rules that are considered to be instinctual.

B) From the Current World Model, derive the underlying Semantic Network of the Primitive Structure Database. This includes both nodes and links.

C) For each node created for the underlying Semantic Network of the Primitive Structure Database, create a Primitive Structure interface node.

Step 3 ¿From the Fuzzy Rule Base and the Semantic Network create the Primitive Structure Database by marking the SNet and applying each pairwise connected set of nodes and their connecting link to the Fuzzy Rule Base, yielding an f-weight.

After Commencing Operation of the Intelligent Machine

Step 4 Create Shadow Coordination Level Petri Net and establish connections to Map Interface Nodes.

While neither ERR-FLAG, nor OP-FLAG nor MC-FAIL is asserted perform Step 5 else perform Step 6:

Step 5 A) Preprocess most likely errored event for each Shadow Coordination Level Petri Net transition and store resulting plan.

B) Monitor introduction of new information into the Current World Model. If applicable perform Step 2 and augment Primitive Structure Database as per Step 3.

Step 6 Using ERR-FLAG, OP-FLAG and MC-FAIL, attempt to identify the error from the information given by the Intelligent Machine. For ERR-FLAG and OP-FLAG, determine the Shadow Coordination Level Petri Net

Error Recovery Generation Algorithm

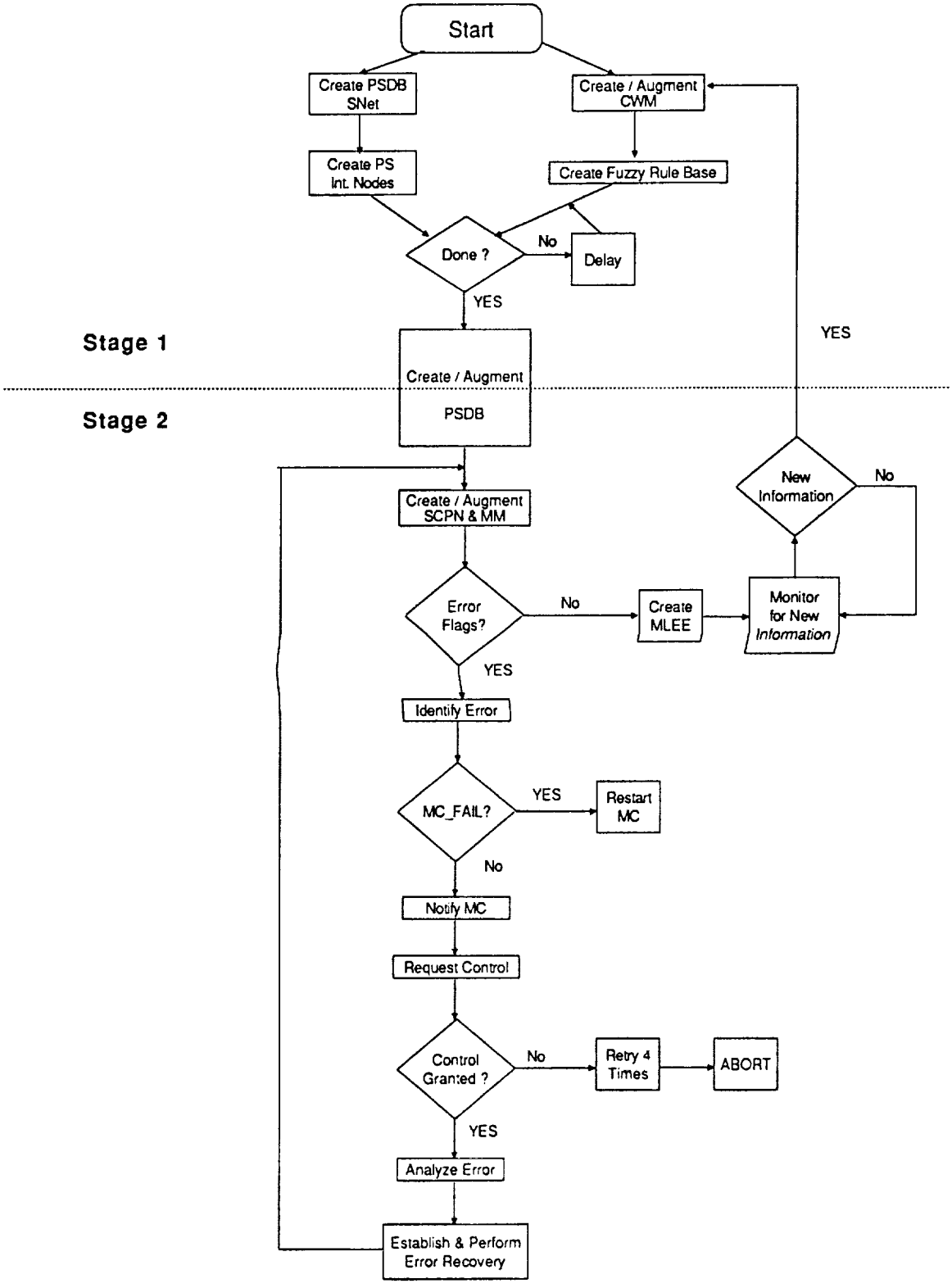

Figure 5

transition from which the error began, and activate Map Interface Node. For MC-FAIL , assume system control and attempt system restoration. If successful, return system control to the Intelligent Machine main controller. If unsuccessful, notify external base for assistance and gracefully shutdown Intelligent Machine operation.

Note: There are several types of errors possible during task execution.

Step 7 Notify the Intelligent Machine's main controller that an error has occurred. Note that the main controller may already know that an error has occurred. This step is to ensure the continuity of error data transmission.

Step 8 A) Request control of the Intelligent Machine's operation from the main controller to prevent interference during error recovery. If not granted, re-try four times. If not granted, abort error recovery.

B) If granted, analyze error information from Step 6. If information is sufficient to enact error recovery, do so. Otherwise use system components (i.e., Vision System, Motion Control System etc.) to try to gain further information on the error type.

Step 9 A) If an error is the same as the calculated most likely error, then execute the preprocessed most likely error, error recovery. If the error is not the most likely error, then establish recovery plans in the plan execution list.

B) Execute first plan in the plan execution list. If successful return control to main controller. If unsuccessful, execute remaining plans in the plan execution list until either all plans are executed or one is successful. If no plans are successful, report irrecoverable error to the main controller and return control to the main controller.

Step 10 Return to Step 5 and continue.

This concludes the Error Recovery Generation Algorithm. The next subsection outlines the System Fault Monitor.

4.2.10 System Fault Monitor

The System Fault Monitor functions to monitor and perform hardware diagnostics of the Planning Coordinator and if desired, the Intelligent Machine to which the Planning Coordinator is connected. Although it performs an error recovery function for hardware, the System Fault Monitor is not one of the major components of the Planning Coordinator. This is because its function is in an area that has been very highly developed. Existing fault diagnostic techniques do suffice.

References

[1] N. Baba. *New Topics in Learning Automata Theory and Applications.* Lecture Notes in Control and Information Science, Springer-Verlag, 1985.

[2] G. Chiola. *GreatSPN 1.5 Software Architecture.* University of Torino, jul 1990.

[3] G. Ciardo. *Manual for the SPNP Package Version3.0*, jun 1990.

[4] F. Dicesare and M.D. Jeng. A review of synthesis techniques for petri nets. *CIM Internal Document, Rensselaer Polytechnic Institute*, 1988.

[5] P. Huber, K. Jensen, and R.M. Shapiro. Hierarchies in colored Petri Nets. In *G. Rozenberg(ed.): Advances in Petri Nets 1990. Lecture Notes in Computer Science*, volume 483, pages 313–341. Springer, Berlin Heidelberg New York, 1990.

[6] I. Koh and F. Dicesare. Modular transformation methods for generalized petri nets and their applications to automated systems. *CIM Internal Document, Rensselaer Polytechnic Institute*, 1988.

[7] B. Kosko. Fuzzy cognitive maps. *International Journal of Man Machine Studies*, 24, January 1986.

[8] B. Kosko. Adaptive inference in fuzzy knowledge networks. In *Proceedings of the ICNN*, 1987.

[9] Don R. Lefebvre and George N. Saridis. A computer architecture for Intelligent Machines. In *Proceeding of IEEE International Conference on Robotics and Automation*, may 1992.

[10] Pedro Lima and Randal Beard. Using neural networks and Dyna algorithm for integrated planning, reacting and learning in systems. Technical Report CIRSSE-122, Center for Intelligent Robotic Systems for Space Exploration (CIRSSE), Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1992.

[11] Pedro U. Lima and George N. Saridis. Measuring Complexity of Intelligent Machines. In *submitted to 1993 IEEE Int. Conf. Robotic Automat.*, may 1993.

[12] John E. McInroy and George N. Saridis. Reliability analysis in Intelligent Machines. *IEEE Transactions on Systems, Man and Cybernetics*, 20(4), 1990.

[13] John E. McInroy and George N. Saridis. Techniques for selecting pose algorithms. *submited to Automatica*, 1992.

[14] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1989.

[15] Y. Narahari and N. Viswanadham. *Stochastic Petri Net Models for Performance Evaluation of Automated Manufacturing Systems.* Elsevier Science Publishers B.V. (North Holland), 1988.

[16] Z. J. Nikolic and K. S. Fu. An algorithm for learning without external supervision and its application to learning control systems. *IEEE Transactions on Automatic Control*, AC-11(13), 1966.

[17] J. E. Peck. A petri net controller for distributed hierarchical systems. Technical Report CIRSSE-109, Center for Intelligent Robotic Systems for Space Exploration (CIRSSE), Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1991.

[18] George N. Saridis. Architectures for Intelligent Machines. Technical Report CIRSSE-58, Center for Intelligent Robotic Systems for Space Exploration (CIRSSE), Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1991.

[19] George N. Saridis and Kimon P. Valavanis. Analytical design of Intelligent Machines. *Automatica*, 24:123–133, 1988.

[20] M.A. Styblinski and B.D. Meyer. Fuzzy cognitive maps, signal flow graphs and qualitative circuit analysis. In *Proceedings of the ICNN*, 1988.

[21] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning in direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, 1992.

[22] Ichiro Suzuki and Tadao Murata. A method for stepwise refinement and abstraction of Petri Nets. *Journal of Computer and Systems Sciences*, 27:51–76, 1983.

[23] J. Traub, G. Wasilkowsky, and H. Wozniakowsky. *Information-Based Complexity*. Academic Press, Inc., 1988.

[24] Kimon P. Valavanis. *A Mathematical Formulation for the Analytical Design of Intelligent Machines*. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1986.

[25] Kimon P. Valavanis and George N. Saridis. Information theoretic modelling of robotic and automation systems. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6):852–872, 1988.

[26] Kimon P. Valavanis and George N. Saridis. Probabilistic modelling of intelligent robotic systems. *IEEE Transactions on Robotics and Automation*, 7(1):164–171, 1991.

[27] R. Valette. Analysis of Petri Nets by stepwise refinements. *Journal of Computer and Systems Sciences*, 18:35–46, 1979.

[28] Fei-Yue Wang and George N. Saridis. A coordination theory for Intelligent Machines. *Automatica*, 26(5):833–844, 1990.

[29] G. Zames. On the metric complexity of causal linear systems, ϵ-entropy and ϵ-dimension for continuous time. *IEEE Transactions on Automatic Control*, AC-24(4):220–230, 1979.

[30] M.C. Zhou and F. DiCesare. Adaptive design of petri net controllers for automatic error recovery. In *Proceedings of the Third IEEE International Conference on Intelligent Control*, August 1988.